

**REMARKS**

The indication of allowable subject matter with respect to claims 4-11 and 22 is appreciated.

**A. Claim 3 was rejected under 35 U.S.C. §102(b) as being anticipated by Rust et al (US 6,223,134). The applicant respectfully traverses this rejection for the following reason(s).**

A class driver of Rust means a driver that can be used in common for groups of products of the measuring equipment, such as an oscilloscope and a DMM (digital multimeter). The class driver of Rust does not connect an application and a driver, as in the DIA (device independent access) hierarchy of the present invention.

- Claim 3 is directed towards a method for commonly controlling device drivers, comprising, in part, the step of:

*arranging a device independent access hierarchy between an application hierarchy and a device driver hierarchy.*

The Examiner states in the rejection (on page 2) "Rust teaches a method for commonly controlling device drivers, comprising the steps of: arranging a device independent access hierarchy between an application hierarchy and a device driver hierarchy. And refers us to Fig. 5, (Class Drivers 304); and col. 13, lines 19-67.

What Rust may or may not teach is subject matter to be considered under §103, not §102.

Referring to Fig. 5 in Rust, Rust discloses, when a user application calls a predetermined

driver, calling the driver according to a pointer with respect to the predetermined driver, whereas the present invention, as claimed, is directed to calling a driver using a common command for drivers.

Under §102, there must be no difference between the claimed invention and the reference disclosure, as viewed by a person of ordinary skill in the field of the invention. *Scripps clinic & Research Foundation v. Genentech, Inc.*, 927 F.2d 1565, 18 USPQ2d 1001, 18 USPQ2d 1896 (Fed. Cir. 1991).

Note that in order for an anticipation rejection to be proper, the anticipating reference **must disclose exactly** what is claimed. "A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." *Verdegaal Bros. v. Union Oil Co. of California*, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987). "The identical invention must be shown in as complete detail as is contained in the ... claim." *Richardson v. Suzuki Motor Co.*, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989). Note here that the Examiner has not relied on "inherency," accordingly, each and every element must be expressly described in Rust.

The underlined portion of the Examiner's statement of rejection above is language not found in Rust, and is in fact the same language used in the Applicant's claim.

A review of Rust finds no use of the term "device independent access."

Accordingly, the rejection is not clear.

Although the Examiner refers us to Fig. 5, (Class Drivers 304); and col. 13, lines 19-67, we cannot determine which of Rust's elements correspond to the term "device independent access" or "device independent access hierarchy."

The Examiner is referred to 37 CFR §1.104(c)(2) which directs the Examiner to designate

the particular part relied on as nearly as practicable, when a reference is complex . The pertinence of each reference, if not apparent, must be clearly explained.

Citing Fig. 5, (Class Drivers 304); and col. 13, lines 19-67 is clearly not as nearly practicable enough since the term "hierarchy" is not used in Rust, and since the term "'device independent access" is not used in Rust.

Additionally, claim 3 is directed towards a method for commonly controlling device drivers, comprising, in part, the step of:

*defining functions available in a corresponding device driver among functions of a function block in a function table.*

According to the present specification, paragraph [0034]: "In the embodiment of the present invention, only functions available in a corresponding device driver among functions of a function block defined and standardized in international organizations such as ITU/RFC (International Telecommunications Union/Request For Comments), *etc.* is defined in a function table. The present invention employs all function blocks defined in standardized documents made by international organizations for standardization such as ITU (International Telecommunications Union), IETE (Internet Engineering Task Force), ETSI (European Telecommunications Standardization Institute), ATM (Asynchronous Transfer Mode) forum, ADSL (Asymmetrical Digital Subscriber Line) forum, *etc.* In the embodiment of the present invention, only functions available in the corresponding device driver among functions of all function blocks defined by the international organizations for standardization are re-defined in the function table."

The Examiner refers us to Rust's col. 6, lines 34-51, IVI engine 306, col. 16, lines 28-49,

"...array of function names...", and col. 23, lines 29-42.

Rust's IVI (interchangeable virtual instrument) engine 306 communicates with each of the class drivers 304, the generic capabilities portion of the specific driver 308 and the instrument-specific capabilities portion of the specific driver 308. The IVI engine also couples to the initialization file referred to as IVI.INI 310. As shown, the IVI engine 306 and the specific driver 308 include bidirectional communication. The IVI engine 306 operates to create and manipulate IVI instrument driver objects. The IVI engine 306 also manages function pointers to the specific driver and calls driver-specific attribute callbacks. The IVI engine 306 further manages all instrument driver attributes and performs Set and Get attribute operations. The IVI engine 306 also performs state caching and attribute simulation operations.

When the user application 302 makes a call or invokes a method on the class driver 304, the class driver 304 accesses information in the IVI Engine 306. The class driver 304 uses services provided by the IVI engine 306 to access the respective function in the specific instrument drivers 308 which performs the operation on the specific instrument. More specifically, when the class driver receives a function call, the IVI engine 306 provides a pointer to the class driver, wherein the pointer points to the respective function in the specific driver. In other words, all calls to the class driver 304 actually invoke the specific instrument drivers 308 of the instrument being called. Thus the class driver 304 itself does not actually configure or "touch" the instrument, but rather preferably in every case the specific instrument drivers 308 of the instrument is invoked to actually perform the configuration or operation. The IVI engine 306 thus operates with a respective class driver 304 to enable operation of the class driver 304 within the system. The IVI engine 306 is capable of

operating with each of the class drivers 304 within the system. Thus the IVI engine 306 is generic to each of the class drivers 304 and provides services to each of the class drivers 304.

The specific drivers 308 is operable to use services and/or functions in the IVI engine 306, such as set attribute and get attribute functions in the IVI engine 306. Likewise, the IVI engine 306 is operable to invoke callbacks in the specific driver 308 to perform its services and/or functions. The IVI engine 306 preferably includes a plurality of set attribute and get attribute functions, each for a respective data type. For example, the IVI engine 306 preferably includes set attribute and get attribute functions for data types such as int32, real 64, Booleans, strings, pointers, etc.

Therefore, the functions, such as the set and get function, of IVI engine 306 are not *functions available in a corresponding device driver*, but instead are only functions of IVI engine 306.

Additionally, there is no *function table* disclosed, and there are no *functions of a function block in a function table* disclosed in Rust, especially with respect to IVI engine 306.

In the columns cited by the Examiner, Col. 23, line 36 mentions a table examined by the IVI engine 306, and this table is located in IVI.ini (a configuration or initialization file) file 310 of Fig. 5.

Rust discloses that the initialization or configuration file (INI file) 310 comprises information on each of the instruments or virtual instruments present in the instrumentation system. The INI file 310 maps Logical Names used in a program to a Virtual Instrument section. The Virtual Instrument Section operates to link a hardware section and a driver section. The Virtual Instrument Section also sets driver modes, such as range checking, simulation, spying, interchangeability checking, and instrument status checking, using parameters called rangeCheck, simulate, spy, interchangeCheck,

and queryInstrStatus. The INI file also defines Virtual Channel Names which map channel name aliases to instrument-specific channel names and specifies a default instrument setup for the virtual instrument.

For example, if the user application generically references a DMM (digital multimeter) as "DMM1", the configuration file stores information regarding which actual instrument and instrument driver in the system corresponds to DMM1. The configuration file stores information regarding the address of the instrument and the location of the instrument driver. This enables the system to load the specific driver and communicate with the instrument.

As an example of the initialization operation, the user application calls the init function in step 422 with the logical name "DMM1". The class driver 304 receives the logical name in step 424 and directs the IVI engine 306 to open up DMM1 in step 426. The class driver 304 also provides an array of function names in step 426 that it expects the instrument driver 308 to have. The IVI engine 306 examines the table in the IVI.ini file in step 428, determines the location of DMM1, and loads the instrument driver for DMM1 in step 430.

The table in the IVI.ini file 310 is not disclosed as being a *function table*, and there is no mention of *functions of a function block in a function table* anywhere in Rust.

In the final rejection readily admits on page 10, line 20 that Rust does not explicitly describe a table.

The Examiner then goes on to state Rust discloses defined functions provided in the specific drivers that are callable by user applications, referring to col. 6, lines 19-51 and col. 14, lines 45-55 and that the specific device driver 308 is called by the IVI Engine 306 to obtain a defined function

from the specific device driver 308, referring to col. 14, lines 52-54 and col. 21, lines 24-30.

Accordingly, as can be seen from the cited sections of Rust, each of the specific instrument drivers 308 are specific to an instrument of a certain class, manufacturer, model and hardware interface type. Thus Rust's system still requires specific drivers 308 for each instrument. Accordingly, even though these functions are looked up in the device drivers 308, meaning that they are stored therein, this does not mean that these function are defined as functions available in a corresponding device driver among functions of a function block in a function table.

Furthermore, the fact that the specific devices drivers have to be accessed in order to call the function of a specific device clearly indicates that Rust's calling function is device dependent, not device independent. Thus Rust fails to disclose *arranging a device independent access hierarchy between an application hierarchy and a device driver hierarchy*, as required by the first feature of claim 3.

Note in Fig. 5 of Rust that there is no element such as a *device independent access hierarchy* disposed between user application 302 and class driver 304. Additionally, IVI Engine 306 is disposed between class drivers 304 and specific driver 308, not between the application 302 and either the class drivers 304 or the specific drivers 308.

More specifically, Rust discloses in col. 6, lines 22+, that the user application preferably makes calls to the class driver. All the functions in the class driver are generic functions, i.e., functions which are common to all or most of the specific drivers within the class. This provides the instrument interchangeability benefits of the present invention. Thus there is clearly there is no element such as a *device independent access hierarchy* disposed between user application 302 and

class driver 304, as evidenced by Fig. 5.

Accordingly, the rejection of claim 3 is deemed to be in error and should be withdrawn.

- Claim 3 is directed towards a method for commonly controlling device drivers, comprising, in part, the step of:

*when a device is initialized, allowing said device independent access hierarchy to generate a device handler identifier ~~based on~~ having a standardized common data format for said device and transmit the generated device handler identifier having the standardized common data format to the application hierarchy of a higher order.*

Here the Examiner refers to several instances wherein the term "handle" appears in Rust, such as col. 20, lines 35-46; col 23, lines 12-21; and col. 24, lines 22-29.

The foregoing feature of claim 3 requires *said device independent access hierarchy to generate a device handler identifier*. In col. 20, lines 35-46, Rust discloses "the user application includes calls to a driver initialization function (init) in the class driver 304 which creates or instantiates the driver and returns a **handle**." There is no *device independent access hierarchy* disclosed in Rust, and it is the class driver 304 which returns a handle.

The cited section col 23, lines 12-21 discusses step 426 in Rust and state: "After step 462 the initialization function call originally made by the user application has been completely executed. From this point on, whenever the user application 302 calls a function in the class driver, the function call includes the handle that is returned from the init function (as discussed above with respect to col. 20, lines 35-46) to specify which instrument is being communicated with." See step



462 in Fig. 7D.

The cited section col. 24, lines 22-29 of Rust summarizes the forgoing.

Additionally, of the above mentioned cited sections of Rust, there is no disclosed device handler identifier generated in Rust *having a standardized common data format*. As can be seen from the Examiner's rejection, the Examiner appears to only be interested in Rust's disclosure of a handle, as there is no mention in the rejection of where Rust is supposed to disclose generation of a device handler identifier *having a standardized common data format*.

In the final rejection, page 10, lines 10, the Examiner states that Rust "teaches these limitations". Besides the fact that Rust fails to teach the foregoing feature of claim 3, as noted previously, the rejection is based on §102, and thus the claimed features must be disclosed, not taught.

Accordingly, the rejection of claim 3 is deemed to be in error and should be withdrawn.

- Claim 3 is directed towards a method for commonly controlling device drivers, comprising, in part, the step of:

*allowing the higher-order application hierarchy to call a predetermined device using the device handler identifier having the standardized common data format, and allowing said device independent access hierarchy to identify a function of the corresponding device driver from the function table using the device handler identifier having the standardized common data format and call the function of the corresponding device driver.*

As noted previously, there is no disclosed *device independent access hierarchy* and there is

no disclosed *function table* in Rust.

Referring to Fig. 5 in Rust, Rust discloses, when a user application calls a predetermined driver, calling the driver according to a pointer with respect to the predetermined driver, whereas the present invention, as claimed, is directed to calling a driver using a common command for drivers.

Therefore, Rust does not **disclose** a method of calling a driver using the common command, but discloses calling the corresponding driver according to an identifier of each driver. Further, Rust teaches that application A and application B respectively call a driver A and driver B **using each identifier**. In the present invention, application A and application B respectively call driver A and driver B through **one command**.

Accordingly, the rejection of claim 3 is deemed to be in error and should be withdrawn.

**B. Claims 1, 2 and 12-21 were rejected under 35 U.S.C. §103(a), as rendered obvious and unpatentable, over Rust et al. in view of Pike et al. (US 6,993,772). The Applicant respectfully traverses this rejection for the following reason(s).**

**Claim 1**

- Claim 1 is directed towards a method for commonly controlling device drivers, comprising, in part, a step of:

*arranging a device independent access hierarchy between an application hierarchy and a device driver hierarchy and applying a standardized rule of said device independent access hierarchy to said application hierarchy and said device driver hierarchy.*

As discussed above, Rust fails to discuss a *device independent access hierarchy* and looking to Fig. 5 in Rust, there is no element such as a *device independent access hierarchy* disposed between user application 302 and class driver 304.

Additionally, Rust fails to discuss a *standardized rule*.

Accordingly, the rejection of claim 1 is deemed to be in error and should be withdrawn.

- Claim 1 is directed towards a method for commonly controlling device drivers, comprising, in part, a step of:

*allowing said application hierarchy and said device driver hierarchy to access the device driver hierarchy and said application hierarchy through the standardized rule of said device independent access hierarchy, respectively.*

The Examiner notes that Rust fails to teach the foregoing feature and refers to Pike's Instrument Engine 102 and Col. 8, lines 3-14. The Examiner then states it would have been obvious to one of ordinary skill the art at the time of the invention was made to combine the teaching of Pike and Rust because the teaching of Pike "would improve the system of Rust by allowing communication between a user application and device driver such that a response could be returned to the user application."

The Examiner has not identified what such a "response" would be nor why it would be of use to Rust.

The Examiner has not identified any problem with Rust's system and method for controlling an instrumentation system such that one of ordinary skill the art at the time of the invention was

made would have desired to look to the related art for a solution to such a system.

The Examiner has not identified how a "response" (noting that we have no idea what the response is supposed be) would improve the system of Rust by allowing communication between a user application and device driver such that a response could be returned to the user application, since communication between a user application and device driver is already allowed such that a handle is already returned from class drivers 304 to user application 302 as disclosed in Rust's col. 20, lines 38-41.

Thus, without a factual basis to support the basis of obviousness, it appears that the Examiner merely speculates that the combination of Pike and Rust would improve the system of Rust.

Deficiencies in the factual basis cannot be supplied by resorting to speculation or unsupported generalities. *In re Warner*, 379 F.2d 1011, 154 USPQ 173 (CCPA 1967) and *In re Freed*, 425 F.2d 785, 165 USPQ 570 (CCPA 1970).

Accordingly, the Examiner fails to establish a *prima facie* bases of obviousness.

*In re Rijckaert*, 28 USPQ2d 1955 (CAFC 1993) states:

"A *prima facie* case of obviousness is established when the teachings from the prior art itself would appear to have suggested the claimed subject matter to a person of ordinary skill in the art." *In re Bell*, 991 F.2d 781, 782, 26 USPQ2d 1529, 1531 (Fed. Cir. 1993) (quoting *In re Rhinehart*, 531 F.2d 1048, 1051, 189 USPQ 143, 147 (CCPA 1976). If the examiner fails to establish a *prima facie* case, the rejection is improper and will be overturned. *In re Fine*, 837 F.2d 1071, 1074, 5 USPQ2d 1596, 1598 (Fed. Cir. 1988).

Additionally, Pike fails to teach or discuss the features of claim 1 noted above as lacking in Rust, such as, *a device independent access hierarchy* and *a standardized rule*.

Accordingly, the rejection of claim 1 is deemed to be in error and should be withdrawn.  
Claim 2 is deemed to be non-obvious for the same reasons as discussed above.

In the final rejection there is no argument provided by the Examiner to dispute the foregoing traversal of the rejection. Instead, the Examiner remarks that the test for obviousness is not whether the features of a secondary reference may be bodily incorporated into the structure of the primary reference; nor is it that the claimed invention must be expressly suggested in any one or all of the references. Rather the test is what the combined teachings of the references would have suggested to those of ordinary skill in the art.

Note here that the Applicant has made no argument concerning whether or not features of Pike could be bodily incorporated into the structure of Rust, nor that the claimed invention must be expressly suggested in any one or all of the references, since that was not the basis of the rejection.

Instead, the Applicant has argued that there has been no *prima facie* showing that there is a problem with the communication between the device driver and user application and there has been no *prima facie* showing that the teaching of Pike "would improve the system of Rust by allowing communication between a user application and device driver such that a response could be returned to the user application." Note that Rust already allows for communication between a user application and device driver such that a response could be returned to the user application.

Accordingly, the rejection is deemed to be in error, and should be withdrawn.

**Claim 12**

- Claim 12 is directed towards a method, comprising, in part, a step of:  
*requesting loss of signal state information based on having a standardized common format by an application to a device independent access hierarchy.*

Looking to both Rust and Pike we find no mention of concern with respect to *loss of signal state information*.

The Examiner refers to Rust's teaching of "check instrument status". However, "check instrument status" is deemed not to correspond to *loss of signal state information*.

Additionally, it is required that the application make the request to the *device independent access hierarchy*. See Applicant's paragraph [0032].

In Rust the "check instrument status" resides in the specific driver 308.

In the final rejection, page 11, lines 13-17, the Examiner refers us to Rust's col. 26, lines 24-33 and col. 36, lines 46-49. Both of these sections clearly disclose that the error checking is a function of the driver. There is no disclosure in these cited sections of Rust to indicate that the error checking is requested by an application nor that there is a request made to a device independent hierarchy.

Accordingly, the rejection of claim 12 is deemed to be in error and should be withdrawn.

- Claim 12 is directed towards a method, comprising, in part, a step of:  
*converting the request from said application into a first device local format and requesting a first device driver to provide the loss of signal state information to said device independent access*

*hierarchy.*

Here the Examiner refers to Rust's col. 15, lines 24-45, and specifically to lines 26-29, *i.e.*, "In other words, when creating a user application 302, the user is generally only required to have knowledge of the class driver 304 . . ."

It is not clear why the Examiner refers to the above cited section of Rust, as there is nothing in col. 15, lines 24-45 dealing with the previous cited portion of Rust regarding "check instrument status".

Similarly, the Examiner refers to Rust's Fig. 8A and col. 24, lines 43-45 which have no connection to Rust regarding "check instrument status".

It appears the Examiner is merely picking and choosing certain parts of Rust in an attempt to deprecate the claims.

It is impermissible within the framework of section 103 to pick and choose from any one reference only so much of it as will support a given position to the exclusion of other parts necessary to the full appreciation of what such reference fairly suggests to one skilled in the art.

*In re Wesslau*, 353 F.2d 238, 241, 147 USPQ 391, 393 (CCPA 1965); see also *In re Mercer*, 515 F.2d 1161, 1165-66, 185 USPQ 774, 778 (CCPA 1975).

One cannot use hindsight reconstruction to pick and choose among isolated disclosures in the prior art to deprecate the claimed invention. *In re Fine*, 837 F.2d at 1075, 5 USPQ2d at 1600.

Pike was not relied on with respect to the foregoing features of claim 12.

Accordingly, the rejection of claim 12 is deemed to be in error and should be withdrawn. The remaining features of claim 12 and claims 13-21 will not be addressed herein, as it should be

quite clear that the combined teachings of Rust and Pike do not support the rejection.

The examiner is respectfully requested to reconsider the application, withdraw the objections and/or rejections and pass the application to issue in view of the above amendments and/or remarks.

Respectfully submitted,



Robert E. Bushnell  
Attorney for Applicant  
Reg. No.: 27,774

1522 K Street, N.W.  
Washington, D.C. 20005  
(202) 408-9040

Folio: P56833  
Date: 9/4/07  
I.D.: REB/MDP